# Python Files Content

modelConfig = {  
 'API\_KEY': 'sk-fvJ6pHWwCEJoy5lyHjUKGd68Q6CRFP0Sv3W6IuoHfYO4i4xr'  
}

import os  
from docx import Document  
  
def add\_to\_word(file\_path, doc):  
 """将文件内容添加到Word文档中"""  
 with open(file\_path, 'r', encoding='utf-8') as file:  
 content = file.read()  
 doc.add\_paragraph(content)  
 doc.add\_paragraph('') # 添加空行作为文件间的分隔  
  
def create\_word\_document(directory, output\_path):  
 """创建Word文档，包含目录中所有.py文件的内容"""  
 # 创建一个新的Word文档  
 doc = Document()  
 doc.add\_heading('Python Files Content', level=1)  
  
 # 遍历目录下的所有文件  
 for root, dirs, files in os.walk(directory):  
 for file in files:  
 if file.endswith('.py'):  
 file\_path = os.path.join(root, file)  
 add\_to\_word(file\_path, doc)  
  
 # 保存Word文档  
 doc.save(output\_path)  
 print(f'Word文档已创建：{output\_path}')  
  
# 设置要扫描的目录和输出文档的路径  
directory\_to\_scan = './'  
output\_document\_path = 'python\_files\_content.docx'  
  
# 创建Word文档  
create\_word\_document(directory\_to\_scan, output\_document\_path)

from langchain\_core.prompts import ChatPromptTemplate  
import json  
  
def extract\_json(json\_str):  
 problem\_data = json.loads(json\_str)  
 return problem\_data  
  
system\_prompt = """  
你是一个中文助手，叫做智知，根据以下context提供内容作答question的题目，尽量不直接给出答案，要引导用户思考，回答给出在answer中  
Question: {question}  
Context: {context}  
History:  
{history}  
"""  
system\_ocr\_prompt = ChatPromptTemplate.from\_messages(  
 [  
 (  
 """  
 query中有一个问题，user\_answer中为用户回答，只能按照```json\n \n```格式输出：   
 回答字段如下：  
 1. answer: 对于query的详细回答  
 2. correct: user\_answer是否正确  
 3. wrong\_place: user\_answer的错误位置  
 4. wrong\_place\_length: user\_answer的错误长度  
 """  
 ),  
 ("human", "{query}"),  
 ("human", "{user\_answer}"),  
 ]  
)  
import json  
  
def parse\_json\_schema(json\_str):  
 try:  
 # 去除字符串两端的```json和```，以及可能的前后空白字符  
 json\_str = json\_str.strip()[7:-3].strip() # 去除开始的```json和结束的```  
   
 # 替换单引号为双引号  
 corrected\_json\_str = json\_str.replace("'", '"')  
 print(corrected\_json\_str)  
 # 解析JSON字符串  
 json\_obj = json.loads(corrected\_json\_str)  
 return json\_obj  
 except json.JSONDecodeError as e:  
 print(f"Error parsing JSON: {e}")  
 return None  
  
# 定义用于构建历史记录的函数  
def build\_history(history):  
 history\_strings = [f"用户输入: {input}, 模型响应: {response}" for input, response in history]  
 return "\n".join(history\_strings)  
  
# 更新创建RAG链的部分  
def create\_rag\_chain\_with\_history(llm, retriever=None, system\_prompt=system\_prompt):  
 def chain(input, history=[]):  
 full\_history = build\_history(history)  
 if(retriever != None):  
 context = retriever.get\_relevant\_documents(input)  
 context\_text = "\n".join([doc.page\_content for doc in context])  
 else:  
 context\_text = ""  
 prompt\_with\_history = system\_prompt.format(question=input,context=context\_text, history=full\_history)  
   
 response = llm(prompt\_with\_history)  
 return response  
  
 return chain  
def create\_ocr\_rag\_chain(llm, system\_prompt=system\_ocr\_prompt):  
 def chain(input, user\_answer = "无"):  
 chain = system\_prompt | llm | parse\_json\_schema  
 print(input,'user', user\_answer)  
 response = chain.invoke({"query": input, "user\_answer": user\_answer})  
 result = {  
 "input": input,  
 "user\_answer": user\_answer,  
 "response": response  
 }  
 return result  
 return chain  
# 构建RAG链  
def create\_rag(llm, retriever=None, system\_prompt=system\_prompt):  
 return create\_rag\_chain\_with\_history(llm, retriever, system\_prompt=system\_prompt)

from fastapi import FastAPI, HTTPException, Request, UploadFile, Depends, File, Form  
from pydantic import BaseModel  
from typing import Dict, List  
import uuid  
from models.moonshot import llm  
from rag import create\_ocr\_rag\_chain, create\_rag  
from retrievers import imageRetriever  
from retrievers.imageRetriever import read\_image\_from\_url, read\_image  
from retrievers.webRetriever import retriever  
from fastapi.middleware.cors import CORSMiddleware  
from sqlalchemy.orm import Session  
from db.index import SessionLocal  
from db.schema import History, OcrHistory, Session, Evaluation, EvaluationData, HistoryListResponse, User  
from fastapi.responses import StreamingResponse  
from auth.schema import UserResponse, UserCreate, TokenResponse  
from firebase\_admin import auth  
from auth.auth import verify\_token, get\_password\_hash, create\_access\_token, verify\_password, get\_current\_user  
import base64  
from graph.index import create\_graph, CustomHumanMessage, execute\_graph  
import os  
  
app = FastAPI()  
IMAGE\_FOLDER = "./images"  
  
origins = ["\*"]  
  
app.add\_middleware(  
 CORSMiddleware,  
 allow\_origins=origins, # 允许的源  
 allow\_credentials=True, # 允许发送Cookie  
 allow\_methods=["\*"], # 允许的方法，如GET、POST等，使用["\*"]来允许所有方法  
 allow\_headers=["\*"], # 允许的请求头，使用["\*"]来允许所有请求头  
)  
session\_histories: Dict[str, List[str]] = {}  
class InputData(BaseModel):  
 input: str  
class OcrData(BaseModel):  
 image: str  
  
def get\_db():  
 db = SessionLocal()  
 try:  
 yield db  
 finally:  
 db.close()  
@app.post("/chat/start-session/")  
async def start\_session(db: Session = Depends(get\_db), user: dict = Depends(get\_current\_user)):  
 session\_id = str(uuid.uuid4())  
 db\_session = Session(id=session\_id)  
 db.add(db\_session)  
 db.commit()  
 db.refresh(db\_session)  
 return {"session\_id": session\_id}  
  
@app.post("/ocr/upload-invoke/")  
async def upload\_file(file: UploadFile = File(...), db: Session = Depends(get\_db), user: dict = Depends(get\_current\_user)):  
 db\_user = db.query(User).filter(User.email == user['email']).first()  
 image\_path = f"./images/{file.filename}"  
 contents = await file.read()  
 with open(image\_path, "wb") as image\_file:  
 image\_file.write(contents)  
 file\_base64 = base64.b64encode(contents).decode("utf-8")  
 questions = imageRetriever.read\_question\_answer\_pair(file\_base64)  
 responses = [create\_ocr\_rag\_chain(llm)(str(question['question']), str(question['user\_answer'])) for question in questions]  
 db\_ocr\_history = OcrHistory(response=str(responses), image\_url=f"/images/{file.filename}", wrong\_place=responses[0]["response"]["correct"], user\_id=db\_user.id)  
 db.add(db\_ocr\_history)  
 db.commit()  
 return {  
 "answer": responses,  
 }  
@app.get("/ocr/history/")  
async def get\_all\_ocr\_history(db: Session = Depends(get\_db), user: dict = Depends(get\_current\_user)):  
 db\_user = db.query(User).filter(User.email == user['email']).first()  
 all\_history = db.query(OcrHistory).all()  
 user\_id = db\_user.id  
  
 # 查询数据库，获取与当前用户ID相关的OCR历史记录  
 user\_ocr\_history = db.query(OcrHistory).filter(OcrHistory.user\_id == user\_id).all()  
 if not all\_history:  
 raise HTTPException(status\_code=404, detail="No OCR history found")  
 return [  
 {  
 "id": history.id,  
 "response": history.response,  
 "image\_url": history.image\_url,  
 "wrong\_place": history.wrong\_place,  
 }  
 for history in user\_ocr\_history  
 ]  
@app.post("/ocr/invoke")  
async def invoke\_ocr(data: OcrData, request: Request, db: Session = Depends(get\_db), user: dict = Depends(get\_current\_user)):  
 db\_user = db.query(User).filter(User.email == user['email']).first()  
 questions = imageRetriever.read\_question\_answer\_pair(read\_image\_from\_url(data.image))  
 responses = [create\_ocr\_rag\_chain(llm)(str(question)) for question in questions]  
 db\_ocr\_history = OcrHistory(response=str(responses), image\_url=data.image, wrong\_place=responses[0]["correct"], user\_id=db\_user['id'])  
 db.add(db\_ocr\_history)  
 db.commit()  
 return {  
 "answer": responses,  
 }  
@app.post("/chat/invoke/")  
async def invoke\_model(data: InputData, request: Request, db: Session = Depends(get\_db), user: dict = Depends(get\_current\_user)):  
 session\_id = request.headers.get("Session-ID")  
 if not session\_id:  
 raise HTTPException(status\_code=400, detail="Invalid or missing Session-ID")  
   
 session = db.query(Session).filter(Session.id == session\_id).first()  
 if not session:  
 raise HTTPException(status\_code=404, detail="Session not found")  
   
 # 获取历史记录  
 history\_records = db.query(History).filter(History.session\_id == session\_id).all()  
 history = [(h.user\_input, h.model\_response) for h in history\_records]  
 state = {"messages": [{"author": "system", "content": "start"}]}  
 # 调用模型链生成响应  
 rag\_chain = create\_rag(llm)  
 graph = create\_graph(rag\_chain)  
 response = execute\_graph(graph, data.input + "".join([h[0] for h in history]))  
   
 # 保存输入和响应  
 db\_history = History(session\_id=session\_id, user\_input=data.input, model\_response=response)  
 db.add(db\_history)  
 db.commit()  
   
 return {  
 "answer": response,  
 "evaluation\_prompt": "Please evaluate the response with a score from 1 to 5, and optionally provide feedback."  
 }  
  
@app.post("/chat/evaluate/", response\_model=None)  
async def evaluate\_model(data: EvaluationData, db: Session = Depends(get\_db), user: dict = Depends(get\_current\_user)):  
 session = db.query(Session).filter(Session.id == data.session\_id).first()  
 if not session:  
 raise HTTPException(status\_code=404, detail="Session not found")  
   
 db\_evaluation = Evaluation(session\_id=data.session\_id, score=data.score, feedback=data.feedback)  
 db.add(db\_evaluation)  
 db.commit()  
   
 return {"message": "Evaluation received successfully"}  
  
@app.get("/chat/history/{session\_id}", response\_model=HistoryListResponse)  
async def get\_history(session\_id: str, db: Session = Depends(get\_db),user: dict = Depends(get\_current\_user)):  
 histories = db.query(History).filter(History.session\_id == session\_id).all()  
 if not histories:  
 raise HTTPException(status\_code=404, detail="No history found for this session")  
 return {"histories": histories}  
  
@app.post("/auth/register/", response\_model=UserResponse)  
def register(user: UserCreate, db: Session = Depends(get\_db)):  
 # Check if the user already exists  
 db\_user = db.query(User).filter(User.email == user.email).first()  
 if db\_user:  
 raise HTTPException(status\_code=400, detail="Email already registered")  
   
 # Create a new user  
 hashed\_password = get\_password\_hash(user.password)  
 new\_user = User(email=user.email, password\_hash=hashed\_password)  
 db.add(new\_user)  
 db.commit()  
 db.refresh(new\_user)  
   
 return {"detail": 'successfully registered'}  
  
  
@app.post("/auth/login/", response\_model=TokenResponse)  
def login(user: UserCreate, db: Session = Depends(get\_db)):  
 # Check if the user exists  
 db\_user = db.query(User).filter(User.email == user.email).first()  
 if db\_user is None or not verify\_password(user.password, db\_user.password\_hash):  
 raise HTTPException(status\_code=401, detail="Invalid credentials")  
  
 # Generate JWT token  
 access\_token = create\_access\_token(data={"sub": user.email})  
   
 return {"access\_token": access\_token, "token\_type": "bearer"}  
  
# 提交个人信息接口  
@app.post("/submit-info/")  
async def submit\_info(  
 nickname: str = Form(None),  
 avatar: UploadFile = File(None),  
 db: Session = Depends(get\_db),  
 userToken: Dict = Depends(get\_current\_user)  
):  
 if not userToken:  
 raise HTTPException(status\_code=401, detail="token not found")  
 user = db.query(User).filter(User.email == userToken["email"]).first()  
 image\_path = f"./images/{avatar.filename}"  
 # 更新昵称和头像  
 if nickname:  
 user.nickname = nickname  
 if avatar:  
 # 处理上传的头像文件  
 avatar\_data = await avatar.read()  
 with open(image\_path, "wb") as f:  
 f.write(avatar\_data)  
 image\_url = f"/images/{avatar.filename}"  
 user.avatar\_data = image\_url  
  
 db.commit()  
 db.refresh(user)  
 return {"id": user.id, "email": user.email, "nickname": user.nickname, "avatar": image\_url}  
  
# 获取个人信息接口  
@app.get("/get-info")  
async def get\_info(db: Session = Depends(get\_db), userToken: Dict = Depends(get\_current\_user)):  
 incorrect\_count = db.query(OcrHistory).filter(OcrHistory.wrong\_place == False).count()  
 user = db.query(User).filter(User.email == userToken["email"]).first()  
 if not user:  
 raise HTTPException(status\_code=404, detail="User not found")  
 return {  
 "id": user.id,  
 "email": user.email,  
 "nickname": user.nickname,  
 "incorrect\_num": incorrect\_count,  
 "avatar\_data": user.avatar\_data  
 }  
   
@app.get("/images/{filename}")  
async def get\_image(filename: str):  
 try:  
 # 构建图片的相对路径  
 image\_path = os.path.join(IMAGE\_FOLDER, filename)  
 # 检查文件是否存在  
 if not os.path.exists(image\_path):  
 raise HTTPException(status\_code=404, detail="Image not found")  
 # 返回图片的StreamingResponse  
 return StreamingResponse(open(image\_path, "rb"), media\_type="image/png")  
 except IOError:  
 raise HTTPException(status\_code=500, detail="Server error")

from langchain\_chroma import Chroma  
from langchain\_community.document\_loaders import WebBaseLoader  
from langchain\_community.embeddings import QianfanEmbeddingsEndpoint  
from langchain\_text\_splitters import RecursiveCharacterTextSplitter  
  
loader = WebBaseLoader(  
 web\_paths=("https://www.zhihu.com/",),  
)  
docs = loader.load()  
  
text\_splitter = RecursiveCharacterTextSplitter(chunk\_size=1000, chunk\_overlap=200)  
splits = text\_splitter.split\_documents(docs)  
vectorstore = Chroma.from\_documents(documents=splits, embedding=QianfanEmbeddingsEndpoint())  
  
# Retrieve and generate using the relevant snippets of the blog.  
retriever = vectorstore.as\_retriever()

import requests  
import json  
import base64  
import urllib.parse as urlparse  
  
with open('zhizhi\_secrete.json') as json\_file:  
 data = json.load(json\_file)  
 client\_id, client\_secret, base\_url = data['baidu\_client\_id'], data['baidu\_client\_secret'], data['baidu\_base\_url']  
  
def get\_access\_token(client\_id, client\_secret):  
 url = f"{base\_url}/oauth/2.0/token?grant\_type=client\_credentials"  
 payload = {  
 'client\_id': client\_id,  
 'client\_secret': client\_secret  
 }  
 headers = {  
 'Content-Type': 'application/x-www-form-urlencoded'  
 }  
 response = requests.request("POST", url, headers=headers, data=payload)  
 return response.json().get('access\_token')  
  
def get\_image\_content\_as\_base64(image\_url):  
 response = requests.get(image\_url)  
 if response.status\_code == 200:  
 return base64.b64encode(response.content).decode("utf8")  
 else:  
 return None  
  
def gene\_resp(access\_token, image\_base64):  
 url = f"{base\_url}/rest/2.0/ocr/v1/general\_basic?access\_token={access\_token}"  
   
 payload = {  
 'image': image\_base64,  
 'detect\_direction': 'false',  
 'detect\_language': 'false',  
 'paragraph': 'false',  
 'probability': 'false'  
 }  
 headers = {  
 'Content-Type': 'application/x-www-form-urlencoded'  
 }  
   
 response = requests.post(url, headers=headers, data=payload)  
 words\_rephrased = [item["words"] for item in response.json().get("words\_result", [])]  
 result = "\n".join(words\_rephrased)  
 return result  
def read\_image(img\_base64=""):  
 access\_token = get\_access\_token(client\_id, client\_secret)  
 if img\_base64:  
 return gene\_resp(access\_token, img\_base64)  
 else:  
 return "Failed to retrieve image."  
def read\_image\_from\_url(image\_url="https://s2.loli.net/2024/09/04/CFlIpU49jMVzqfo.png"):  
 img\_base64 = get\_image\_content\_as\_base64(image\_url)  
 read\_image(img\_base64)  
   
def read\_question\_answer\_pair(image\_base64=get\_image\_content\_as\_base64('https://s2.loli.net/2024/09/08/73a2sJchuVzgbTC.png')):  
 access\_token = get\_access\_token(client\_id, client\_secret)  
 url = f"{base\_url}/rest/2.0/ocr/v1/doc\_analysis?access\_token={access\_token}"  
   
 payload = {  
 'image': image\_base64,  
 'detect\_direction': 'false',  
 'detect\_language': 'false',  
 'paragraph': 'true', # 设置为true以便将文本段落分组  
 'probability': 'false'  
 }  
 headers = {  
 'Content-Type': 'application/x-www-form-urlencoded'  
 }  
   
 response = requests.post(url, headers=headers, data=payload)  
 results = response.json().get("results", [])  
   
 # 初始化结果数组  
 output\_results = []  
   
 # 用于暂存问题和手写回答的变量  
 current\_question = ""  
 current\_answer = ""  
 for item in results:  
 word = item["words"]["word"]  
 words\_type = item["words\_type"]  
  
 if words\_type == "print":  
 if not current\_answer: # 如果当前问题不为空，则保存上一个问题和答案  
 current\_question += word  
 else:  
 output\_results.append({"question": current\_question, "answer": current\_answer})  
 current\_question = word # 重置问题  
 current\_answer = ""  
 elif words\_type == "handwriting":  
 current\_answer += word + " "   
 # 检查是否有最后一个问题和答案未被添加  
 if current\_question or current\_answer:  
 output\_results.append({"question": current\_question.strip(), "user\_answer": current\_answer.strip()})  
 # print(str(output\_results))  
 return output\_results  
if \_\_name\_\_ == '\_\_main\_\_':  
 print(read\_question\_answer\_pair())

import jwt  
from datetime import datetime, timedelta  
from typing import Optional  
from passlib.context import CryptContext  
from fastapi import HTTPException, Header  
  
pwd\_context = CryptContext(schemes=["bcrypt"], deprecated="auto")  
  
# 配置密钥和算法  
SECRET\_KEY = "your\_secret\_key"  
ALGORITHM = "HS256"  
  
def create\_access\_token(data: dict, expires\_delta: Optional[timedelta] = None):  
 to\_encode = data.copy()  
 if expires\_delta:  
 expire = datetime.now() + expires\_delta  
 else:  
 expire = datetime.now() + timedelta(minutes=15)  
 to\_encode.update({"exp": expire})  
 encoded\_jwt = jwt.encode(to\_encode, SECRET\_KEY, algorithm=ALGORITHM)  
 return encoded\_jwt  
  
def verify\_token(token: str):  
 try:  
 payload = jwt.decode(token, SECRET\_KEY, algorithms=[ALGORITHM])  
 return payload  
 except jwt.ExpiredSignatureError:  
 return None  
 except jwt.PyJWTError:  
 return None  
   
def get\_password\_hash(password: str):  
 return pwd\_context.hash(password)  
  
def verify\_password(plain\_password: str, hashed\_password: str):  
 return pwd\_context.verify(plain\_password, hashed\_password)  
  
def get\_current\_user(authorization: str = Header(...)):  
 if not authorization.startswith("Bearer "):  
 raise HTTPException(status\_code=401, detail="Invalid authorization header")  
   
 token = authorization[len("Bearer "):]  
 payload = verify\_token(token)  
 if payload:  
 return {"email": payload["sub"]}  
 else:  
 raise HTTPException(status\_code=401, detail="Invalid or expired token")

from pydantic import BaseModel  
  
class UserCreate(BaseModel):  
 email: str  
 password: str  
  
class UserResponse(BaseModel):  
 detail: str  
  
class TokenResponse(BaseModel):  
 access\_token: str  
 token\_type: str

from pydantic import BaseModel  
from langgraph.graph import StateGraph, END, MessagesState  
from typing import Annotated, TypedDict, List  
from langchain\_core.messages import BaseMessage, HumanMessage  
import operator  
  
class UserMessage(BaseModel):  
 input: str  
  
class CustomHumanMessage(HumanMessage):  
 def \_\_init\_\_(self, content: str, author: str = 'system', \*\*kwargs):  
 super().\_\_init\_\_(content, \*\*kwargs)  
 self.author = author  
  
def merge\_additional\_info(current, update):  
 current.update(update)  
 return current  
# 创建状态图  
def create\_graph(fn):  
 graph\_builder = StateGraph(MessagesState)  
 def get\_content(state: MessagesState):  
 messages = state["messages"]  
 if not messages: # 确保消息列表不为空  
 return provide\_approach(state)  
  
 # 获取最后一条消息  
 last\_message = messages[-1]  
 # 确保最后一条消息是 CustomHumanMessage 类型的实例  
 if isinstance(last\_message, CustomHumanMessage):  
 last\_message\_content = last\_message.content.lower() # 获取最后一条消息内容并转换为小写  
 else:  
 last\_message\_content = last\_message.get('content', '').lower() # 默认为空字符串  
 return last\_message\_content, messages  
   
 def judge\_need(state: MessagesState):  
 last\_message\_content, messages = get\_content(state)  
 # 检查关键词或短语  
 if "详细解释" in last\_message\_content or "不清楚" in last\_message\_content or "为什么" in last\_message\_content:  
 return "provide\_detailed\_explanation"  
 # 检查问题复杂性或其他条件  
 elif "如何" in last\_message\_content and len(messages) > 1: # 如果问题包含"如何"且不是第一次询问  
 return "provide\_detailed\_explanation"  
 # 默认不需要详细解释  
 else:  
 return "provide\_approach"  
   
 def provide\_approach(state: MessagesState):  
 # 提供问题解决思路  
 last\_content, messages = get\_content(state)  
 print(last\_content)  
 return {"messages": [CustomHumanMessage(content=f"存在如下问题：{last\_content}，给出这个问题的简洁思路，引导思考，不给出答案")]}  
  
 def provide\_detailed\_explanation(state: MessagesState):  
 # 提供深入解释  
 last\_content, messages = get\_content(state)  
 return {"messages": [CustomHumanMessage(content=f"存在如下问题：{last\_content}，关于这个问题的详细解释是...")]}  
 def call\_llm(state: MessagesState):  
 # 提供深入解释  
 last\_content, messages = get\_content(state)  
 print(last\_content)  
 return {"messages": [CustomHumanMessage(content=fn(last\_content))]}  
 def end(state: MessagesState):  
 # 结束  
 last\_content, messages = get\_content(state)  
 return {"messages": [CustomHumanMessage(content=last\_content)]}  
 def choose\_direction(state: MessagesState):  
 last\_content, messages = get\_content(state)  
 return {"messages": [CustomHumanMessage(content=last\_content)]}  
  
 # 添加节点到图中  
 graph\_builder.add\_node("judge\_need", choose\_direction)  
 graph\_builder.add\_node("provide\_approach", provide\_approach)  
 graph\_builder.add\_node("provide\_detailed\_explanation", provide\_detailed\_explanation)  
 graph\_builder.add\_node("call\_llm", call\_llm)  
 graph\_builder.add\_node("end", end)  
  
 # 添加边  
 graph\_builder.add\_edge("provide\_approach", "call\_llm")  
 graph\_builder.add\_edge("provide\_detailed\_explanation", "call\_llm")  
 graph\_builder.add\_edge("call\_llm", "end")  
 graph\_builder.add\_conditional\_edges("judge\_need", judge\_need)  
 # 设置入口点  
 graph\_builder.set\_entry\_point("judge\_need")  
  
 # 编译图  
 graph = graph\_builder.compile()  
 return graph  
  
def execute\_graph(graph, user\_input: str):  
 state = {"messages": [CustomHumanMessage(content=user\_input, author='human')]}  
 response = graph.invoke(state)  
 return response["messages"][-1].content

from langchain\_community.llms.moonshot import Moonshot  
import os  
  
os.environ["MOONSHOT\_API\_KEY"] = "sk-NoXsbhonxcroaTvbzOPPqDOIkqsp9FervmPE5rCGkbwR2H6V"  
os.environ["QIANFAN\_AK"] = "GFPPuWQ2XgPyB6o3FsUqmUSU"  
os.environ["QIANFAN\_SK"] = "nXhmp2CjXIhU5fxTTGT0fGx6nYyHuFY8"  
  
llm = Moonshot(model="moonshot-v1-128k")

from sqlalchemy import create\_engine, Column, String, Integer, Text  
from sqlalchemy.orm import declarative\_base  
from sqlalchemy.orm import sessionmaker  
  
USERNAME = 'root'  
PASSWORD = 'Ak472700'  
PORT = '3306'  
DB\_NAME = 'Zhizhi'  
  
DATABASE\_URL = f"mysql+pymysql://{USERNAME}:{PASSWORD}@localhost:{PORT}/{DB\_NAME}"   
  
engine = create\_engine(DATABASE\_URL)  
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)  
Base = declarative\_base()

from sqlalchemy import Column, Integer, String, Text, ForeignKey, BLOB, LargeBinary  
from sqlalchemy.orm import Mapped, mapped\_column, relationship  
from typing import List  
import sys  
import os  
from pydantic import BaseModel  
  
# 获取当前脚本的目录  
current\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_))  
  
# 获取父级目录  
parent\_dir = os.path.dirname(current\_dir)  
  
# 将父级目录添加到 sys.path  
sys.path.append(parent\_dir)  
from db.index import Base, engine  
  
class User(Base):  
 \_\_tablename\_\_ = "users"  
  
 id = Column(Integer, primary\_key=True, index=True)  
 email = Column(String(36), unique=True, index=True)  
 password\_hash = Column(String(255))  
 nickname = Column(String(255), nullable=True)  
 avatar\_data = Column(String(255), nullable=True)  
  
  
class Session(Base):  
 \_\_tablename\_\_ = "sessions"  
  
 id = Column(String(36), primary\_key=True, index=True) # 为 String 类型指定长度  
 history = relationship("History", back\_populates="session")  
 evaluations = relationship("Evaluation", back\_populates="session")  
  
# 会话历史模型  
class History(Base):  
 \_\_tablename\_\_ = "histories"  
  
 id = Column(Integer, primary\_key=True, index=True)  
 session\_id = Column(String(36), ForeignKey("sessions.id")) # 同样为此处的 String 类型指定长度  
 user\_input = Column(Text)  
 model\_response = Column(Text)  
 session = relationship("Session", back\_populates="history")  
  
  
class OcrHistory(Base):  
 \_\_tablename\_\_ = "ocr\_histories"  
  
 id = Column(Integer, primary\_key=True, index=True)  
 image\_url = Column(Text)  
 wrong\_place = Column(Text)  
 response = Column(Text)  
 user\_id = Column(Integer, ForeignKey("users.id"))  
# 评价模型  
class Evaluation(Base):  
 \_\_tablename\_\_ = "evaluations"  
  
 id: Mapped[int] = mapped\_column(Integer, primary\_key=True, index=True)  
 session\_id: Mapped[str] = mapped\_column(String(36), ForeignKey("sessions.id"))  
 score: Mapped[int] = mapped\_column(Integer)  
 feedback: Mapped[str] = mapped\_column(Text)  
  
 session = relationship("Session", back\_populates="evaluations")  
  
class HistoryResponse(BaseModel):  
 id: int  
 session\_id: str  
 user\_input: str  
 model\_response: str  
  
 class Config:  
 orm\_mode = True  
  
  
class HistoryListResponse(BaseModel):  
 histories: List[HistoryResponse]  
  
class EvaluationData(BaseModel):  
 session\_id: str  
 score: int # 评价分数，如1到5  
 feedback: str # 用户的文字反馈  
  
  
Base.metadata.create\_all(bind=engine)